Distributed QMC Algorithms: New Strategies for and Performance Evaluation

EXTENDED ABSTRACT

Laurentiu Cucos and Elise de Doncker

Keywords. Quasi-Monte Carlo, synchronous/ asynchronus distributed integration, lattice rules, computational finance, multivariate t distribution, algorithm analysis.

Abstract

We compare alternative strategies of a distributed algorithm to integrate a multidimensional function using Quasi-Monte Carlo techniques. We give a parallel performance analysis and experimental results, including near ideal speedup results for a computational finance problem in 360 dimensions and for a 5-dimensional t-distribution arising in multiple comparison problems, using up to 30 processors of a Beowulf type cluster.

1 Introduction

We apply a Quasi-Monte Carlo (QMC) method which computes a sequence of randomized lattice rules to aproximate integrals over a d-dimensional hypercube region [3, 2]. Let

$$K_N(\boldsymbol{\beta}) = \frac{1}{N}\sum_{i=1}^{N} f(\{\frac{i}{N}\mathbf{v} + \boldsymbol{\beta}\}), \qquad (1)$$

where $\mathbf{v}$ is the (Korobov) lattice generator vector, randomized by a uniformly distributed random vector β [1]. Then with a sample set (of size q) of random $\boldsymbol{\beta}_j$, the mean

$$\bar{K}_N = \frac{1}{q}\sum_{j=1}^{q} K_N(\boldsymbol{\beta}_j) \qquad (2)$$

allows for a standard error estimation by

$$E_N^2 = \frac{1}{q(q-1)}\sum_{j=1}^{q}(K_N(\boldsymbol{\beta}_j) - \bar{K}_N)^2. \qquad (3)$$

The algorithm calculates the $\bar{K}_N$ values of (2) for successively larger numbers N of integration points until either an answer is found to the user-specified accuracy or the function count limit is reached.

We denote by k_{ij} a cell of work $K_{N_i}(\boldsymbol{\beta}_j)$, where $i \in [1, r_{\max}]$, $j \in [1, q]$, Currently we have $r_{\max} = 25$ with $(N_1, \ldots, N_{25}) = (31, \ldots, 601942)$ [4], and generally $N_i >> N_j$, for $i > j$. The ith row of cells is $R_i = \{k_{ij} \mid j \in [1, q]\}$.

The error generally decreases in a stepwise manner from one row to the next. The error estimate is expected to improve along a row.

2 Single Cell Algorithm

Consider solving this problem on p processors, one of which is assigned a controlling role, the remaining ones being workers. In the single cell version of the algorithm, currently used in PARINT1.1, each of the workers gets assigned one cell of work k_{ij} at a time. The controller updates the result and sends a new cell if neccessary [2]. Since the computations are asynchronous, different processors may be working on different rows. At termination it is not necessary that a row is completed in order to use its result (if the row has reached a set minimum length).

Results from this algorithm tend to show a degradation in performance when the number of workers is relatively large (compared to q). Subsequently we attribute this behavior to a general occurrence of *break-*

ing loss, which is loss in parallel performance caused by processors doing useless work towards the end of the computation. This is actually replaced by *starvation loss*, where processors are idle waiting for work, if the computation proceeds to the last possible row (of index $N_{\max}$).

We consider the special case where the execution in fact behaves in a synchronous manner and the number of processors p is a multiple of the number of cells in a row, that is, $p = tq$ with $t > 0$.

Assume the problem requires n rows to terminate. Initially, the first q processors are working on row R_1, the next q processors work on R_2 and so on. Assuming that the processors finish their cells and receive new tasks at the same time, then once R_1 is done, the first set of q processors moves on to R_{t+1}; once R_2 is done the second set starts row R_{t+2} and so on. When the last row required for this problem is assigned, all other rows of lower index are either completed or in the process of being completed. Since the work required for lower index rows is much smaller than the work required for R_n, and there are only q processors assigned to R_n, all other workers are going to be reassigned to rows R_l, $l > n$ before R_n is completed. However this work is useless, since the controller stops the computation when R_n is completed; thus breaking loss is incurred.

In order to evaluate the parallel performance, let τ be the time required to complete a function evaluation; then the sequential computing time $T_1 = q\sum_{i=1}^{n} N_i\tau$. In the case under consideration, the total parallel execution time equals the time of the processors that work on the last row R_n. That is,

$$T_p = \sum_{s=1}^{r} N_{k_s}\tau, \tag{4}$$

where $r = \lceil \frac{nq}{p} \rceil$, $k_s = k_1 + t(s-1)$, $k_1 = n - (r-1)t$, and $k_r = n$. In this case the theoretical speedup is

$$S = \frac{T_1}{T_p} = q\frac{\sum_{i=1}^{n} N_i}{\sum_{s=1}^{n/t} N_{k_s}}. \tag{5}$$

For example, with $q = 6, p = 30, n = 25$, 5 gives $S = \frac{6\sum_{i=1}^{25} N_i}{N_5+N_{10}+N_{15}+N_{20}+N_{25}} = 15.628$. Measuring the runtime experimentally (on a homogeneous cluster), we obtain a speedup of 15.541.

Note that the total communication time is small as only one pair of messages per cell is required, thus amounting to $\mathcal{O}(nq)$.

Now let us consider $p = q$ $(t = 1)$. All the processors work in one row at a time. Once a row is finished all workers move to the next one. When the last row is completed there are no processors working on cells beyond the last row, resulting in a near ideal speedup. The same can be shown if $p < q$ and q is a multiple of p (however, breaking or starvation loss may be incurred when q is not a multiple of p).

Note that this method imposes a limitation on the number of processors, since the maximum number of workers to participate in the computation is nq (corresponding to the case when each worker gets assigned only one cell). Since usually q is generally taken in the range [4,16], this limitation starts for relatively small numbers of processors.

The above observations lead to the idea to distribute every row among all the workers, so that (in a synchronous setting) all workers at a time process the same row.

3 Row Distributed Algorithm

This strategy splits every cell k_{ij} into p pieces k_{ij}^m $m \in [1,p]$ of equal size, where every individual piece has N_i/p points to compute. Every worker w is assigned a work unit $u_i^w = \{k_{ij}^w \mid j = 1,\ldots,q\}$. After the controller has received the row results from all workers, it assembles the cell results $k_{ij} = \sum_w k_{ij}^w$, and generates the row mean and standard error estimate according to (2) and (3).

This new strategy tends to eliminate breaking loss considerably. Even though the number of messages is now bounded by $\mathcal{O}(np)$ and the size of a result message has increased by a factor q, the net time increase is not that pronounced since communications at one worker is generally overlapped by computation performed at other workers. The apparent limitation of the number of workers to the size of the first rule (31), where each worker gets only q function evaluations, can be alleviated by skipping the first row (or rows as needed). An issue that remains is that the performance is easely degraded if some processors are slower than others, making this strategy unsuitable for heterogeneous systems.

To overcome this problem, we consider modifying the approach by assigning fixed size work units to a processor regardless of the row it is working on. This will split the early rows into fewer work units, and

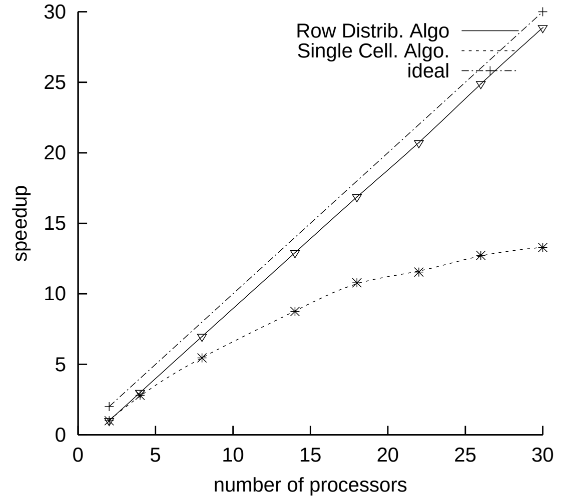

Figure 1: Speedup for Finance Problem

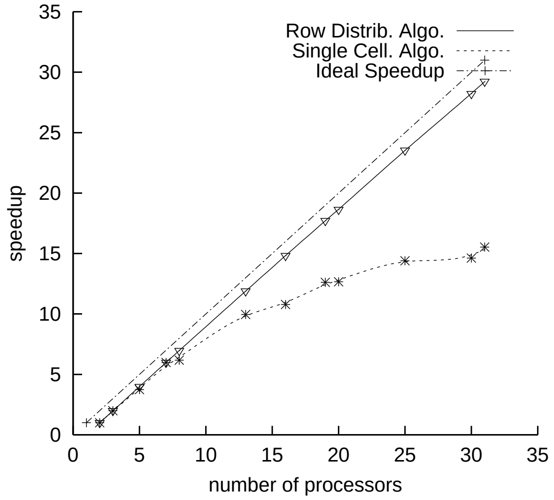

Figure 2: Speedup for MVT Problem

later rows in possibly more units than the number of workers. The work unit size for different processors can be pro-rated according to the processor speed; or can be determined by the controller based on round-trip time during the early part of the execution.

4 Experimental results

In this section we present results from parallel runs on our Beowulf cluster composed of 32 (1.2GHz) Athlon processors connected by a fast ethernet switch. This is our largest size homogeneous subcluster.

Using the mortgage-backed security problem from [6, 7], we tested the new algorithm for up to 30 processors. Figure 1 shows the speedup obtained when calculating the (360-dimensional) integral. The function count limit was set to 17,552; that corresponds to $n = 7$, with 8 cells per row. A function evaluation requires 0.3 sec. The result obtained was 286045.51, with an estimated error of 7.391.

We also include results for a multivariate Student-t distribution problem arising in multiple comparisons [5, 2]. Figure 2 shows the speedup obtained for a 5-dimensional integral. The function limit was set to 21,666,096 corresponding to 25 rows, using 6 cells per row. The result obtained was 0.899999 with an estimated error of 9.46e-6. A function evaluation requires 13μsec.

The new algorithm exhibits a near ideal speedup for problems of different computational granularity, up to the maximum number of processors used.

5 Conclusions

We presented a highly scalable distributed algorithm to integrate multidimensional functions using QMC techniques on homogeneous systems. Its superior performance is justified theoretically and verified experimentally.

For future work we consider an adaptation of the algorithm for heterogeneous environments, by providing each processor with fixed size work units throughout the course of the computation. The work unit size may depend on processor characteristics.

References

[1] Cranley, R., and Patterson, T. N. L. Randomization of number theoretic methods for multiple integration. *SIAM J. Numer. Anal. 13* (1976), 904–914.

[2] de Doncker, E., Cucos, L., Zanny, R., and Genz, A. Parallel computation of the multivariate t-distribution. In *Proceedings of the High Performance Computing Symposium (HPC'01)* (2001), pp. 129–134.

[3] de Doncker, E., Zanny, R., Ciobanu, M., and Guan, Y. Distributed Quasi Monte-Carlo methods in a heterogeneous environment. In *Proceedings of the IPDPS Heterogeneous Computing Workshop* (2000), pp. 200–206.

[4] Genz, A. MVNDST: Software for the numerical computation of multivariate normal probabilities, 1998. Available from web page at http://www.sci.wsu.edu/math/faculty/genz/homepage.

[5] Genz, A., and Bretz, F. Numerical computation of multivariate t-probabilities with application to power calculation of multiple contrasts. *J. Stat. Comp. Simul. 63* (1999), 361–378. http://www.sci.wsu.edu/math/faculty/genz/homepage.

[6] Ninomya, S., and Tezuka, S. Toward real-time pricing of complex financial derivatives. *Applied Mathematical Finance 3* (1996), 1–20.

[7] Paskov, S. H. Computing high dimensional integrals with applications to finance. Tech. rep., Columbia University, Department of Computer Science, 1994. CUCS-023-94.